

Adv DevOps Case Study

My case study topic is as follows:-

Infrastructure as Code with Terraform (case study number 3)

- Concepts Used: Terraform, AWS S3, and EC2.
- Problem Statement: "Use Terraform to provision an AWS EC2 instance and an S3 bucket. Deploy a sample static website on the S3 bucket using the EC2 instance as the backend server."
- Tasks:
 - Write a Terraform script to create an EC2 instance and an S3 bucket.
 - Deploy the static website on the S3 bucket.
 - Use the EC2 instance to interact with the S3 bucket and log the actions.

Topic introduction:

Infrastructure as Code (IaC) is a DevOps practice that automates the provisioning and management of cloud infrastructure using code, rather than manual processes. Terraform, an open-source tool developed by HashiCorp, is widely used to implement IaC, enabling the creation, modification, and versioning of infrastructure efficiently across different cloud platforms. In this case study, Terraform is used to provision an AWS EC2 instance and an S3 bucket, while integrating these resources to deploy and manage a static website. This approach reduces human error and increases the scalability and consistency of infrastructure management.

Significance of Using Terraform in AWS

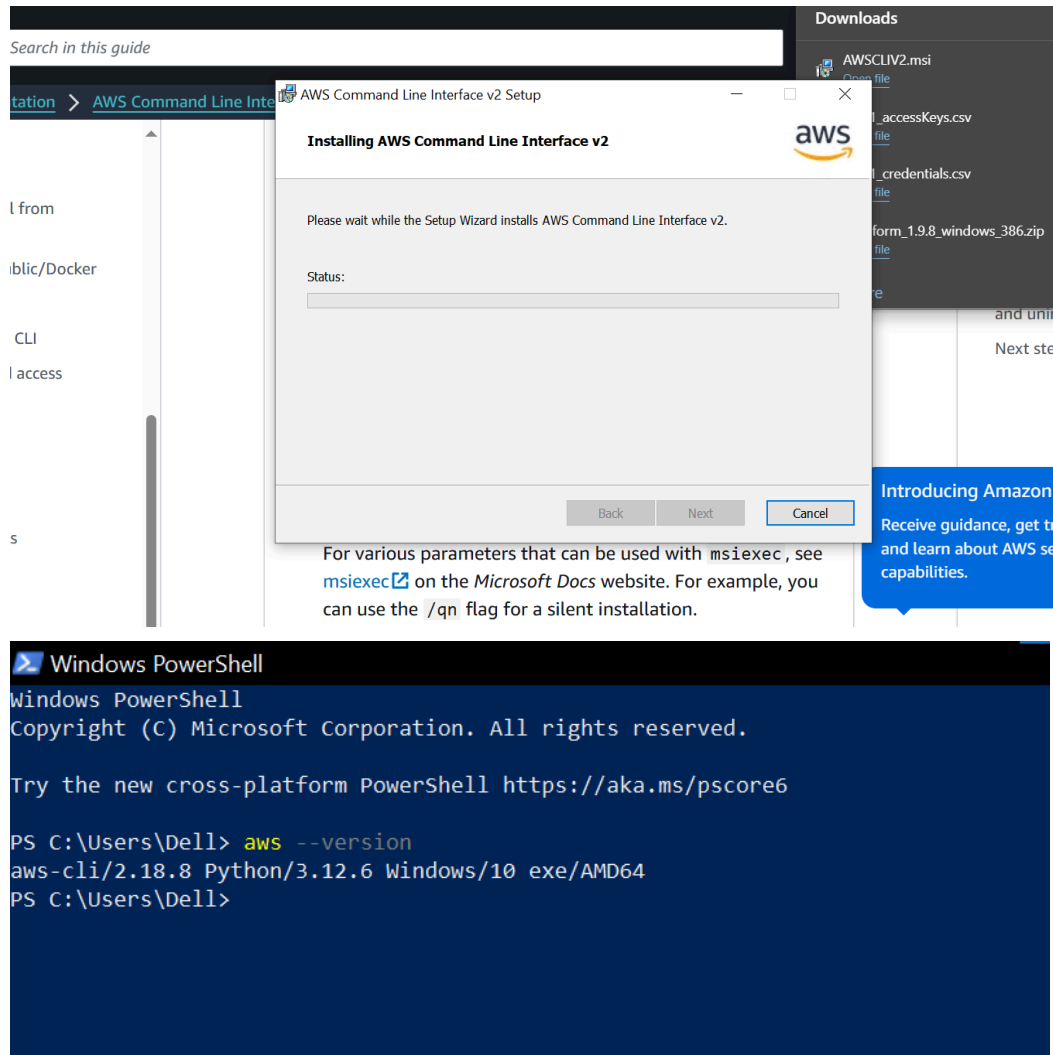
1. Automation: With Terraform, cloud resources can be created, modified, or destroyed automatically, reducing the need for manual intervention.
2. Version Control: The infrastructure code can be versioned and maintained in repositories, enabling teams to track changes, roll back configurations, and collaborate effectively.
3. Consistency: Infrastructure can be provisioned consistently across different environments, reducing configuration drift and ensuring that development, testing, and production environments are identical.
4. Multi-Cloud Support: Terraform provides the flexibility to manage resources not only on AWS but also across multiple cloud providers with a single tool.

Integration with SEMESTER-5 mini project:

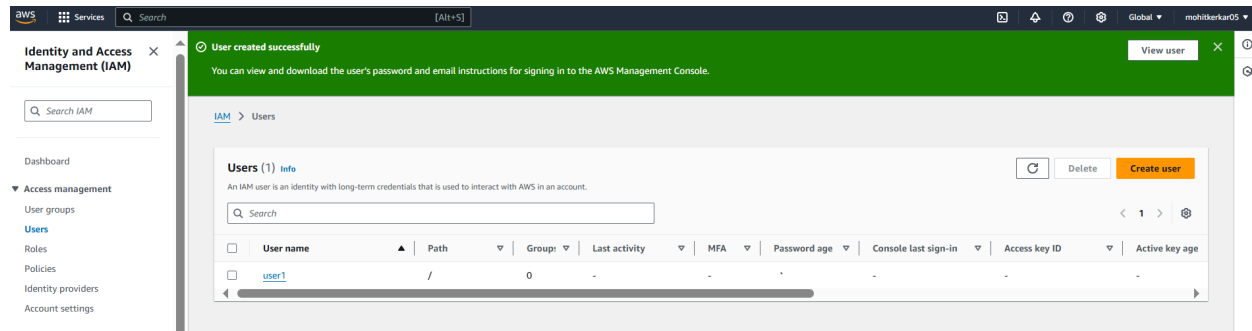
My project for this semester is a Venue booking application. Now, it involves a ton of features like allowing customers to book, leaving feedback, searching and sorting venues according to ratings, budget and a number of other things. I feel that the involvement of Terraform and the communication between ec2 instance and s3 bucket would largely be relevant when the application is to be deployed and existing features require new additions or refinement. The S3 bucket holding the application's deployment would communicate effectively with our ec2 instance for the deployment of a new module, new feature or discarding an existing obsolete feature from our deployed application.

Step by Step procedure to perform the assigned task is as follows:-**Step 1: AWS CLI installation on local machine**

This is done to avail the services of AWS on our local terminal/machine

**Step 2: IAM user and access key creation**

This step is key to using Terraform commands to access and create resources/services onto our AWS account. Creation of the IAM user with a certain set of permissions (preferably administrator rights) further leads to providing a gateway to our local machine to remotely access and use or create resources as per predefined constraints. The IAM user access key id and the shared key are used to locate our AWS account so as to further work with it



By performing 'aws configure' command, we current aws user to be accessed from our local machine to be whatever credentials we have.

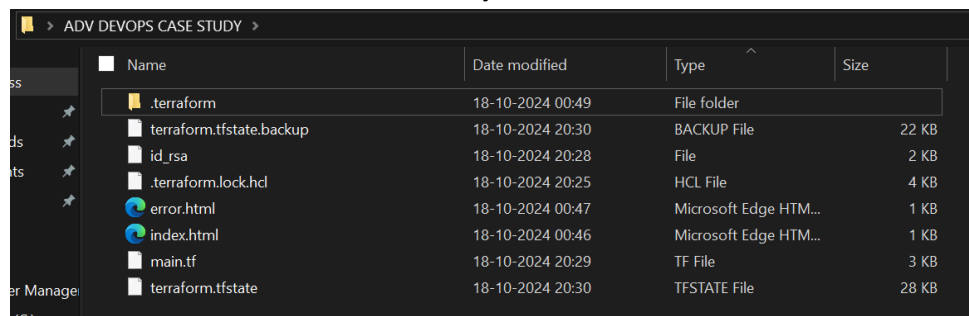
```
PS C:\Users\Dell> aws configure
AWS Access Key ID [*****W3NK]:
AWS Secret Access Key [*****7ltf]:
Default region name [us-east-1]:
Default output format [json]:
PS C:\Users\Dell> █
```

Step 3: File creation

In order to create an EC2 instance, an S3 bucket containing two html files, we need terraform to build the infrastructure for these resources/services. We do that by creating a directory of our choice and creating a file with '.tf' extension within it. This '.tf' extension allows us to write our 'IaC' code in HashiCorp Configuration Language.

Now, i have created 2 files named **index.html** and **error.html** to be put in my S3 bucket which i will create

And i have created a file named **main.tf** which will write the declarative code for my ec2 and s3 infrastructure to be created in the way that we intend.



Step 4: Defining main.tf and using terraform

main.tf file should consist of the following contents so as to fulfill our tasks.

This code must contain:

1. IAM user connection using access key id
2. EC2 instance and bucket configuration

3. Code snippets for connecting index.html and error.html
4. Code snippets for creating a separate public and private key

Save this file, navigate to the current directory inside cmd or powershell and run the following commands

1. Terraform init

```
C:\Users\De11\Desktop\ADV DEVOPS CASE STUDY>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.72.1...
- Installed hashicorp/aws v5.72.1 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

2. Terraform plan

```
C:\Windows\System32\cmd.exe

+ ebs_block_device (known after apply)
+ enclave_options (known after apply)
+ ephemeral_block_device (known after apply)
+ instance_market_options (known after apply)
+ maintenance_options (known after apply)
+ metadata_options (known after apply)
+ network_interface (known after apply)
+ private_dns_name_options (known after apply)
+ root_block_device (known after apply)
}

# aws_key_pair.my_key will be created
+ resource "aws_key_pair" "my_key" {
+   name           = (known after apply)
+   fingerprint    = (known after apply)
+   id             = (known after apply)
+   key_name       = "id_rsa_mohit_09122004"
+   key_name_prefix = (known after apply)
+   key_pair_id    = (known after apply)
+   key_type       = (known after apply)
+   public_key     = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCG/1t7FAHAlg/Xohw2YhtkX5Kc8juU7jOWE8P72kz1Efb0de1Tb1d5UFDXP8DXR5Kdw9Wp188gf2P31IdFvPFatni6GAQhy2iunvo3p3u
obaso3v2LW6kapicBpIimgumAYh+7gRavEYgeM/uheF5jKQ+41psqveRH0gcSol2Vt4AX6qs4R5yFd1zE2Z+8m73JmnDn6S8QKPGVtnRd4ZQwAqtkF2pwRz3JugR0uG98fkoEnx2Z2VR5ev/Hd1MEP6Ab1eUnF6k4HI+qRVhMjg
m/8DimpIhr17ROK/KKXPPUuUSHFxpKB35/Iha38F10CDAnDzBpQcUK+k31DRZ5"
+   tags_all       = (known after apply)
}

Plan: 2 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
```

3. Terraform apply

```
C:\Windows\System32\cmd.exe

+ network_interface (known after apply)
+ private_dns_name_options (known after apply)
+ root_block_device (known after apply)
}

# aws_key_pair.my_key will be created
+ resource "aws_key_pair" "my_key" {
+   name           = (known after apply)
+   fingerprint    = (known after apply)
+   id             = (known after apply)
+   key_name       = "id_rsa_mohit_05122004"
+   key_name_prefix = (known after apply)
+   key_pair_id    = (known after apply)
+   key_type       = (known after apply)
+   public_key     = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCG/1tZFAHug/XoWiaZYHTXK5kc8juU7jOWEBP72kzLEfDbDeiTb1dSUFDP8XDXR5Kdw9WqP188gF2P31IdfVFPqtni6GAQhX2iuvvo3p3u
obaso3v2UN6kqpiCBPimgum4Vh+7gRavEYgeM/uhf5JKQ+41psqveRHQc5oL2Vt4AX6qs4R5yFd1zEZ+8m73Jmndn6S8DKPGVtnRd4ZQhWagtKF2pRz3JugR0uG9BFkoEnXZ2VR56v/HdIMEP6Ab1eUnF6k4HI+qRVHfJq
m/0Dmp3hr17ROX/KXFPPLUwJHPxP4825/4h438PjCCdAnDz8p9UcUK+K3LDRZ5"
+   tags_all       = (known after apply)
}

Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

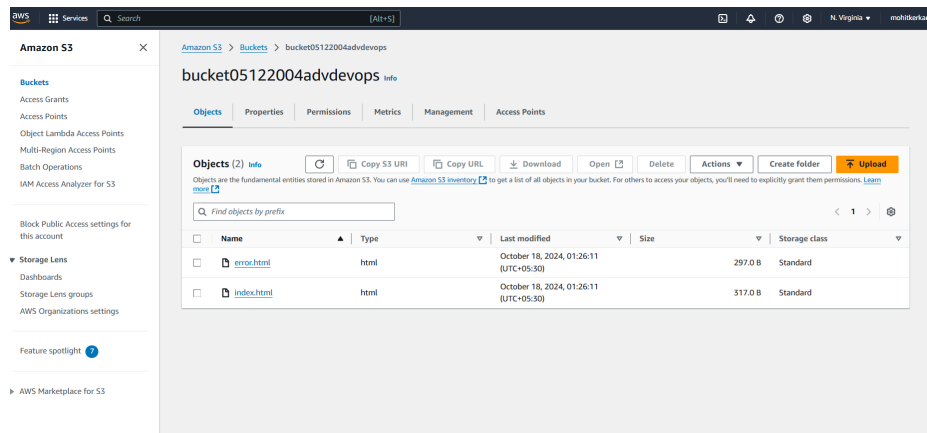
Enter a value: yes

aws_key_pair.my_key: Creating...
aws_key_pair.my_key: Creation complete after 1s [id=id_rsa_mohit_05122004]
aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 16s [id=i-0fab2a7b6562aab64]

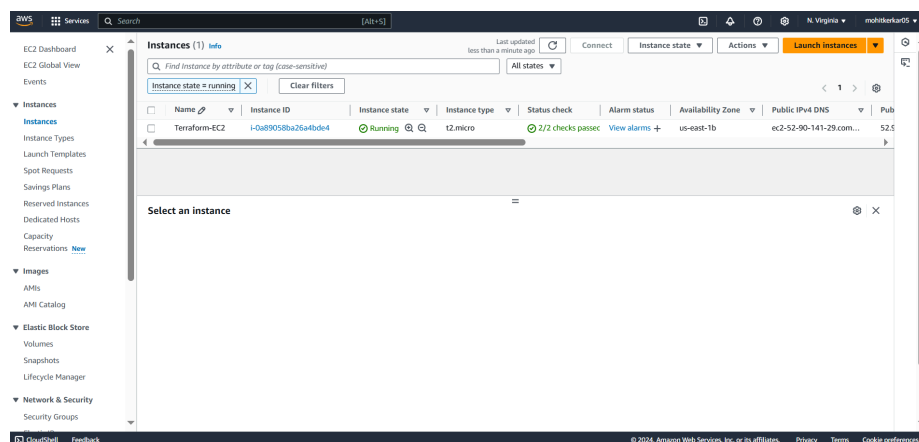
apply complete! Resources: 2 added, 0 changed, 0 destroyed.

C:\Users\De1\Desktop\ADV DEVOPS CASE STUDY>
```

Successful running of these commands indicates that our ec2 instance is up and running and our S3 bucket is successfully created and the 2 html files are deployed within it S3 bucket:



EC2 instance:



Along with these things, we also have generated a public and a private key for accessing our instance locally.


We will use the path of our private key which is generated in the same directory to remotely login onto our ec2 instance using ssh

```
PS C:\Users\Dell> ssh -i "C:\Users\Dell\Desktop\ADV DEVOPS CASE STUDY\id_rsa" ec2-user@ec2-54-236-238-111.compute-1.amazonaws.com
The authenticity of host 'ec2-54-236-238-111.compute-1.amazonaws.com (54.236.238.111)' can't be established.
ED25519 key fingerprint is SHA256:EenS2neDHnrkMDN0Vw3UrqmIf9uKvgvfY7iiM+KYHic.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-236-238-111.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
```

The terminal window shows the command prompt at PS C:\Users\Dell>. The user runs the command ssh -i "C:\Users\Dell\Desktop\ADV DEVOPS CASE STUDY\id_rsa" ec2-user@ec2-54-236-238-111.compute-1.amazonaws.com. The output displays a warning about the host's authenticity, showing the ED25519 key fingerprint as SHA256:EenS2neDHnrkMDN0Vw3UrqmIf9uKvgvfY7iiM+KYHic. The user responds with 'yes' to continue the connection. A final warning message states: 'Warning: Permanently added \'ec2-54-236-238-111.compute-1.amazonaws.com\' (ED25519) to the list of known hosts.'

Step 5: Showing interaction between ec2 instance and S3 bucket

Make a new text file



adv devops case study text.txt - Notepad

File Edit Format View Help

Hello from Mohit!

Making a case study for Adv DevOps Lab


Really understanding the flow of things now ...

Open new terminal and run this command to upload our newly made text file on our instance

```
scp -i "path_to_your_key.pem" "path_to_local_file"
```

ec2-user@your-ec2-public-ip:/home/ec2-user/.

This command is used to upload a random text file onto our instance



The screenshot shows a Windows PowerShell terminal window with the following text:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Dell> scp -i "C:\Users\Dell\Desktop\ADV DEVOPS CASE STUDY\id_rsa" "C:\Users\Dell\Desktop\adv devops case study text.txt" ec2-user@ec2-54-236-238-111.compute-1.amazonaws.com:/home/ec2-user/
adv devops case study text.txt                                100% 108      0.3KB/s   00:00
PS C:\Users\Dell>
```

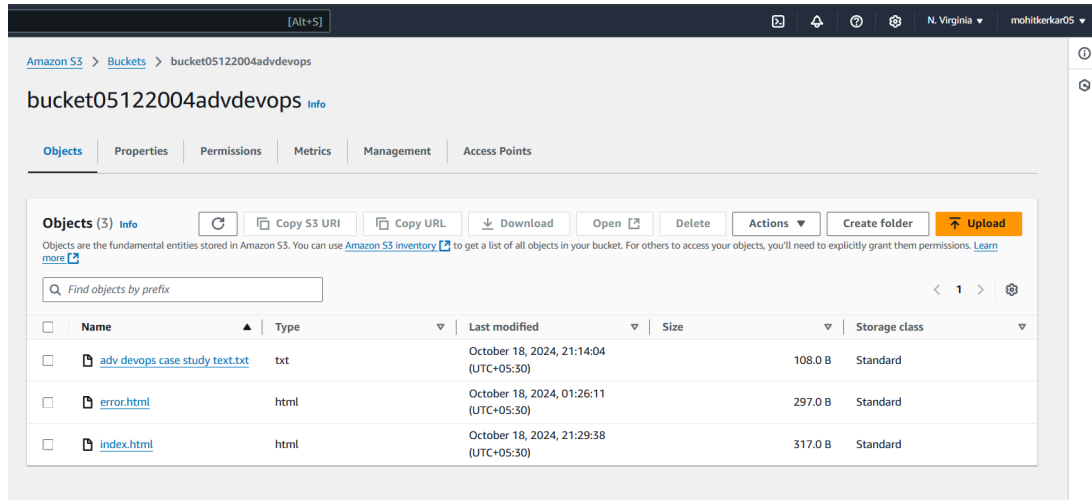
Then upload that file onto our bucket from the original terminal and list them just for confirmation

Command to upload the file onto our bucket: **aws s3 cp /home/ec2-user/you.txt**

s3://your-s3-bucket-name/

```
[ec2-user@ip-172-31-45-106 ~]$ aws s3 cp "adv devops case study text.txt" s3://bucket05122004advdevops/
upload: ./adv devops case study text.txt to s3://bucket05122004advdevops/adv devops case study text.txt
[ec2-user@ip-172-31-45-106 ~]$ aws s3 ls s3://bucket05122004advdevops/
2024-10-18 15:44:04      108 adv devops case study text.txt
2024-10-17 19:56:11      297 error.html
2024-10-17 19:56:11      317 index.html
```

In this image, we can confirm that the upload is indeed successful and the communication between our instance and the bucket is taking place flawlessly.



Use this command in the original terminal to log the actions of the S3 bucket.

echo "Uploaded file 'filename.txt' to S3 on \$(date)" >> action_log.txt

```
[ec2-user@ip-172-31-45-106 ~]$ echo "Downloaded text file from S3 on $(date)" >> action_log.txt
[ec2-user@ip-172-31-45-106 ~]$ cat action_log.txt
Downloaded text file from S3 on Fri Oct 18 15:54:13 UTC 2024
```

Conclusion: Through this case study, I learned how to effectively use Terraform to automate cloud infrastructure setup on AWS, specifically provisioning EC2 instances and S3 buckets. I gained practical experience in writing Terraform scripts, configuring AWS resources, and establishing interaction between EC2 and S3 for file management. Additionally, I developed a deeper understanding of Infrastructure as Code principles, the importance of automation in DevOps, and how tools like Terraform can simplify infrastructure management while ensuring consistency and scalability.